06191 Abstracts Collection
Rigorous Methods for Software Construction and Analysis
— Dagstuhl Seminar —

Jean-Raymond Abrial[1] and Uwe Glässer[2]

[1] ETH Zürich, CH
jabrial@inf.ethz.ch
[2] SFU Burnaby, CA

Abstract. From 07.05.06 to 12.05.06, the Dagstuhl Seminar 06191 "Rigorous Methods for Software Construction and Analysis" was held in the International Conference and Research Center (IBFI), Schloss Dagstuhl. During the seminar, several participants presented their current research, and ongoing work and open problems were discussed. Abstracts of the presentations given during the seminar as well as abstracts of seminar results and ideas are put together in this paper. The first section describes the seminar topics and goals in general. Links to extended abstracts or full papers are provided, if available.

Keywords. Formal Methods, Program Verification, Abstract State Machines, Event-B

06191 Executive Summary – Rigorous Methods for Software Construction and Analysis

We survey here the key objectives and the structure of the Dagstuhl Seminar 06191, which was organized as Festkolloquium on the occasion of Egon Börger's 60th birthday, in May 2006 in Schloss Dagstuhl, Germany.

Keywords: Executive Summary

Joint work of: Abrial, Jean-Raymond; Glässer, Uwe

Extended Abstract: http://drops.dagstuhl.de/opus/volltexte/2006/665

A Light Simulator-Verificator for Timed Abstract State Machines

The talk is a short description of main features of a portable simulator-verificator for reactive timed Abstract State Machines. The time may be continuous or discrete, the time constraints are linear (extendable to polynomial ones), the reaction may be instantaneous or with non deterministic bounded delays. The construction of runs goes on together with the verification of a formula in a First Order Times Logic. There are simple languages for the generation of inputs, for resolving non determinism and for displaying the results. The implementation is being done in Java.

Keywords: Abstract State Machine, real time, reactive system, simulation, verification, First Order Timed Logic

Joint work of: Slissenko; Anatol; Vassiliev, Pavel

Using formally defined design patterns to improve system developments.

Jean-Raymond Abrial (ETH Zürich, CH)

The concept of "design pattern" is well known in Object Oriented Technology. The main idea is to have some sort of reproducible engineering micro-design that the software designer can use in order to develop new pieces of software.

In this presentation, I try to borrow such OO ideas and incorporate them within the realm of formal methods. First, I will proceed by defining (and prove) two formal patterns, where the second one happens to be a refinement of the first one. As a matter of fact, one very often encounters such patterns in the development of reactive systems, where several chains of reactions can take place between an environment and a software controller, and vice-versa. Second, the patterns are then used many times in the development of a complete reactive system where several pieces of equipment interact with a software controller. It results in a systematic construction made of six refinements. The entire system is proved completely automatically. The relationship between the formal design patterns and the formal development of the problem at hand will be shown to correspond to a certain form of refinement.

I think that such an approach to formal developments can be generalized fruitfully to other patterns. It results in very concise and systematic developments.

Keywords: Design pattern

A High-Level Specification for Mediators (Virtual Providers)

Michael Altenhofen (SAP - Karlsruhe, D)

We define a high-level model to mathematically capture the behavioural interface of abstract Virtual Providers (VP), their refinements and their composition into rich mediator structures. We show for a Virtual Internet Service Provider example how to use such a model for rigorously formulating and proving properties of interest.

Joint work of: Altenhofen, Michael; Börger, Egon; Lemcke, Jens

An Empirical Pilot Study Investigating the Cost of Maintaining Specifications: Software Cost Reduction versus Real-time UML

Daniel M. Berry (University of Waterloo, CA)

Specifying a computer-based system (CBS) in a formal specification language can reduce downstream, development errors and rework, but the resulting formal specifications are considered expensive to write and to maintain, discouraging their adoption. With a single-subject experiment this paper explores the costs of modifying specifications written in two different specification languages: a Parnas-Table specification language, Software Cost Reduction (SCR), and a state-of-the-practice specification language, Real-Time Unified Modeling Language (RT-UML). In the experiment, the subject specified two different CBSs, (1) a bidirectional formatter (BDF) and (2) a bicycle computer (BC), with each specification language, but in the opposite order, to balance an expected learning effect arising from specifying the same problem twice with different specification languages. Later in the experiment, the requirements for each CBS were modified, and the subject modified each specification to reflect the changes. The subject recorded (1) the number of hours he needed to write and update each specification, (2) the number of defects he inserted into each specification, and (3) the number of hours the subject needed to repair these defects.

The results show that the cost to modify a specification are highly dependent on both the problem and the specification language used.

There is no evidence that a specification written in a Parnas-Table specification language is easier to modify than a specification written in state-of-the-practice specification language.

Keywords: SCR, RT-UML, pilot study, modifying specifications

The Inevitable Pain of Software Development: Why There Is No Silver Bullet

Daniel M. Berry (University of Waterloo, CA)

Programming technology has improved immensely since its earliest days.

However, no single improvement can be classified as a silver bullet, despite all claims of its proponents. A vexing question is why there has been no silver bullet. A variety of programming improvements, i.e., models, methods, artifacts, and tools, are examined to determine that each has a step that programmers find painful enough that they habitually avoid or postpone the step. This pain is generally where the programming accident meets the essence of software and its relentless volatility. Hence, there is no silver bullet. It is claimed no substantial programming improvement can avoid all pain; therefore there will never be a silver bullet.

Keywords: Programming methods, programming technology, silver bullet, pain

See also: Berry, D.M., “The Inevitable Pain of Software Development: Why There Is No Silver Bullet,” M. Wirsing, S. Balsamo, A. Knapp (eds.), Monterey 2002, Radical Innovation of Software and Systems Engineering in the Future, Workshop Proceedings, pp. 28-47, Venice, Italy, 7-11 October 2002.

Formal Methods, the Very Idea, Some Thoughts

Daniel M. Berry (University of Waterloo, CA)

The paper defines formal methods (FMs) and describes economic issues involved in their application. From these considerations and the concepts implicit in “No Silver Bullet”, it becomes clear that FMs are best applied during requirements engineering. A theory of why formal methods work when they work is offered and it is suggested that FMs help the most when the applier is most ignorant about the problem domain.

Keywords: Formal methods, why they work, requirements engineering, importance of ignorance

See also: Berry, D.M., “Formal Methods, the Very Idea, Some Thoughts on Why They Work When They Work,” Science of Computer Programming, 42:1, 11-27, January, 2002.

Ten Commandments: Ten Years Later: Lessons for ASM, B, Z and VSR-net

Jonathan P. Bowen (London South Bank Univ. - London, GB)

Just over a decade ago, a paper "Ten Commandments of Formal Methods" suggested some guidelines to help ensure the success of a formal methods project. It proposed ten important requirements (or "commandments") for formal developers to consider and follow, based on our knowledge of several industrial application success stories, most of which have been reported in more detail in two books. The paper was surprisingly popular, is still widely referenced, and used as required reading in a number of formal methods courses. However, not all have agreed with some of our commandments, feeling that they may not be valid in the long-term. We re-examine the original commandments ten years on, and consider their validity in the light of a further decade of industrial best practice and experiences, especially with respect to formal notations like ASM, B and Z. We also cover the activities of the UK Verified Software Repository Network (VSR-net) in the context of Grand Challenge 6 on Dependable Systems Evolution.

Keywords: Formal methods; software engineering

Joint work of: Bowen, Jonathan P.; Hinchey, Michael G.

Full Paper:
http://doi.ieeecomputersociety.org/10.1109/MC.2006.35

See also: Bowen, Jonathan P. and Hinchey, Michael G., Ten Commandments of Formal Methods Ten Years Later, IEEE Computer, 39(1):4048, January 2006.

Characterizing Event-B models as ASMs

Egon Börger (Università di Pisa, I)

To facilitate the communication between the B community and the ASM community, we shortly characterize basic Abstract State Machines (ASMs, for the B community) and in terms of basic ASMs the semantical core of event-B models (for the ASM community).

Keywords: Event-B, ASM

A new IEEE 1394 leader election protocol

Dominique Cansell (LORIA - Nancy, F)

The IEEE 1394 tree identify protocol illustrates the adequacy of the event-driven approach used together with the B Method. This approach provides a complete framework for developing mathematical models of distributed algorithms. A specific development is made of a series of more and more refined models. Each model is made of a number of static properties (the invariant) and dynamic parts (the guarded events). The internal consistency of each model as well as its correctness with regard to its previous abstraction are proved with the proof engine of Atelier B, which is the tool associated with B. In the case of IEEE 1394 tree identify protocol, the initial model is very primitive: it provides the basic properties of the graph (symmetry, acyclicity, connectivity), and its dynamic parts essentially contain a single event which elects the leader in one shot. Further refinements introduce more events, showing how each node of the graph non-deterministically participates in the leader election. At some stage in the development, message passing is introduced. This raises a specific potential contention problem, whose solution is given. The last stage of the refinement completely localises the events by making them take decisions based on local data only.

Keywords: Abstract model; B method; Event-driven approach; Proof-based development; Proof engine; Refinement

Joint work of: Abrial, Jean-Raymond; Méry; D.

Jasmine - Accessing Java code from CoreASM

Antonio Cisternino (Università di Pisa, I)

CoreASM allows execution of ASM specifications; it has a micro-kernel architecture allowing extensions, even to the language grammar, by external plugins. Jasmine is a CoreASM plugin whose goal is to expose into the CoreASM world Java objects. This allows ASM specifications to benefit of the rich set of classes available on the Java platform. The plugin introduces a new background into the CoreASM and the access to Java objects is always mediated by the plugin preserving the mathematical purity of the CoreASM execution model. Java objects are not part of the machine state, but they are considered part of the environment. The plugin extends CoreASM syntax with constructs for object creation, method invocation, and field access. It also deal with type conversions needed to marshal CoreASM types into their Java countepart and vice-versa. Jasmine allows specifications to drive real pieces of software; it can be used for black-box testing, to provide GUI to control specifications, and other applications. The interface is two-way so that also Java code can benefit of the CoreASM runtime.

Keywords: ASM, CoreASM, Java, Reflection

Joint work of: Cisternino, Antonio; Gervasi, Vincenzo; Farahbod, Roozbeh

CoreASM: An Extensible ASM Execution Engine

Roozbeh Farahbod (Simon Fraser University, CA)

The CoreASM project was first introduced in ASM 2005 as a novel attempt to make abstract state machines (ASMs) executable. The aim of this project is to specify and implement an extensible ASM execution engine for an extensible language that is as close as possible to the mathematical definition of pure ASMs. We present the architecture of the CoreASM engine together with its extensibility mechanisms that are used by the engine to accommodate arbitrary backgrounds, rule forms, and scheduling policies. An alpha implementation of the engine and some standard plug-ins will also be presented.

Keywords: Abstract State Machines, CoreASM, Executable Specifications

Joint work of: Farahbod, Roozbeh; Gervasi, Vincenzo; Glaesser, Uwe; Memon, Mashaal

Modeling the .NET CLR Exception Handling Mechanism for a Mathematical Analysis

Nicu Georgian Fruja (ETH Zürich, CH)

We provide a complete ASM model for the exception handling mechanism of the Common Language Runtime (CLR), the virtual machine underlying the interpretation of .NET programs. The model fills some gap in the ECMA standard for CLR and is used to sketch the exception handling related part of a soundness proof for the CLR bytecode verifier.

Keywords: Exception handling, type safety, CIL bytecode, CLR, .NET, ASM

Joint work of: Fruja, Nicu Georgian; Börger, Egon

Full Paper:
http://www.jot.fm/issues/issue_2006_04/article1

See also: Modeling the .NET CLR Exception Handling Mechanism for a Mathematical Analysis

A Distributed ASM-Theorem

Andreas Glausch (HU Berlin, D)

Six years ago, Gurevich defined the class of "sequential algorithms" by help of only a few, intuitively convincing and amazingly general requirements. Despite their evident simplicity, the expressiveness of sequential algorithms exceeds classical computational models. Gurevich proved that sequential algorithms are covered by the computational model of "Sequential Abstract State Machines" (sequential ASMs). During the last years, this result has been extended to various variants of ASMs including nondeterministic, parallel, and interactive versions.

In the spirit of these results, I present a class of "distributed algorithms" defined by likewise intuitive requirements. To this end, I introduce "actions" as locally confined changes of a state.

A distributed run then represents a partial order of action occurrences. I present a theorem stating that this class of distributed algorithms is covered by a distributed computation model derived from sequential ASMs.

Keywords: Abstract State Machines, concurrency, distributed semantics

Full Paper:
http://www2.informatik.hu-berlin.de/top/download/publications/techreport196.pdf

See also: Andreas Glausch and Wolfgang Reisig. Distributed Abstract State Machines and Their Expressive Power. Informatik-Berichte 196, Humboldt-Universität zu Berlin, January 2006.

Feasibility in Event-B

Stefan Hallerstede (ETH Zürich, CH)

Event-B is suitable for various modelling problems, in particular, sequential program development. A fundamental property to be proved initially for sequential programs development is feasibility, i.e. whether the initially specified post-condition is satisfiable under the stated pre-condition.

There is an "obvious" way of proving feasibility at the beginning of a sequential program development, because, in Event-B, each event is associated with a feasibility proof obligation. It would be possible to use this for proving feasibility of sequential programs. But the corresponding proof usually requires instantiating an existential quantifier and this might be sometimes problematic.

However, there is a better way of doing it that suits well with refinement. The algorithm to be constructed is supposed to establish the post-condition, too. Thus, we can save the initial feasibility proof by exploiting this fact.

In this presentation, we formally justify this approach and describe a corresponding refinement method for the development of sequential programs in Event-B. An example illustrates our approach.

Keywords: Feasibility, Event-B

Animating and Model Checking B Specifications with Higher-Order Recursive Functions

Michael Leuschel (Universität Düsseldorf, D)

Real-life specifications often contain complicated functions. Animation and validation of such functions and specifications is very important. However, such functions pose a major challenge to animation and model checking.

Earlier versions of ProB required that functions be explicitly expanded which is prohibitively expensive or impossible. The central idea of this new research is to compile such functions into symbolic closures which are only examined when the function is applied to some particular argument. This enables ProB to successfully animate and model check a new class of specifications, where animation is especially important due to the involved nature of the specification. We will illustrate this new approach on an industrial case study.

Keywords: B-Method, Model Checking, Animation, Logic Programming, Visualization

Joint work of: Leuschel, Michael; Bendisposto, Jens

Extended Abstract: http://drops.dagstuhl.de/opus/volltexte/2006/640

Designing a Proof tool for Engineering use

Farhad Mehta (ETH Zürich, CH)

This talk address some of the difficulties of using proof tools in an engineering setting. In particular it focuses on some of the pains engineers face when using proof tools and suggests ways to make the act of proving more palatable. These suggestions are the basic design decisions for the new Event-B prover tool used in the RODIN platform.

Keywords: Formal proof, interactive proof tools, computer based systems engineering

Using wp-like (hence B-like) reasoning for security and privacy

C. Carroll Morgan (Univ. of New South Wales, AU)

I will discuss how security and privacy can be reasoned about in what looks very like the ordinary wp- (hence also B-) style. The state is separated into "visible-" and "hidden" parts, using extra declarations, and then a modified refinement relation is introduced that "preserves ignorance" of the hidden part.

There are certain broad "structuring principles" that should apply if the method is to scale-up, and I'll discuss what they are. As an example, I will derive The Dining Cryptographers protocol via a program algebra based on the restricted refinement relation, showing therefore that we haven't restricted refinement so much that soundness is retaind simply by being unable to refine anything at all. The fetchingly named "Refinement Paradox" crops up along the way, and we'll see how it's not such a problem after all.

With any luck, B- and similar "refinement engines" might one day be able to carry out such proofs semi-mechanically.

Keywords: Refinement, security, privace, predicate transformers, logic of knowledge

Data Abstraction in Spec# and Boogie

Peter Müller (ETH Zürich, CH)

Data abstraction allows specifications to express program behavior in an implementation-independent way. Such abstract behavior specifications are important to support subtyping and information hiding. In languages like Eiffel, the Java Modeling Language (JML), and Spec#, where contracts are based on side-effect free expressions of the programming language, data abstraction is expressed via model fields and side-effect free methods. In this talk, I present a sound and modular verification methodology for model fields, discuss its generalization to method calls in specifications, and identify open problems for future work.

Keywords: Verification, contracts, object-oriented programming

Relaxing B restrictions on invariant composition using Spec# approach

Marie-Laure Potet (LSR - IMAG, F)

One of the major aspects of the B method is its notion of component (abstract machine, refinement and implementation) which can be proved and developed in an independent way. Invariant can be stated relatively to the variables and operations set defined in a given component. Proof obligations are made at the level of component definition and the compositionality principle ensures invariant preservation in any admissible B compositions. Compositionality is based on a set of rules which control how components are used and combined, resulting in that in no further proof obligations. In this way invariant composition is a transparent process for developers as soon as they respect B restrictions.

On the other hand the Spec# approach is based on the dynamic notion of "ownership" which characterizes which object instance can constrain, through an invariant, others object instances. The Spec# approach is based on a set of meta-invariants which describe in which point invariants are guaranted, depending on the ownership relation.

This approach is a very flexible one but the counterpart is that developers must master invariant compositionality through meta-invariants and meta-instructions allowing them to locally invalidate or validate invariants. A solution is to use meta-schemas (as design patterns), which are pre-labelled by information about invariant validity, corresponding to classical architecture forms. For instance the B restrictions can be seen as meta-schemas which ensure, by construct, invariant preservation in all points outside the concerned operations.

The aim of the talk is to show how the actual architectural restrictions of the B method can be interpreted as a particular case of Spec# ownership relation and to propose some extensions to make these restrictions less strict, without too complicated overheads for developers. Finally we study how invariant composition and refinement can interact.

Keywords: B method, Spec#, invariant composition, refinement

Joint work of: Boulmé, Sylvain; Potet, Marie-Laure

Integration of ASM tools using Metamodels

Andreas Prinz (Agder University College - Grimstad, N)

There are several tools for execution, analysis and verification of Abstract State Machines. They all differ in their version of ASMs they handle, in the syntax, the permitted constructs and maybe even in the semantics. This way it is not possible to use them together on the same models.

In this presentation it is shown that the integration between different ASM tools can be done using meta-models. The solution includes the handling of differences between the meta-models.

Keywords: Tool Integration, ASM, Modelling

Exploiting the ASM method within the Model-driven Engineering paradigm

Elvinia Riccobene (Universitá di Milano, I)

Model-driven Engineering (MDE) is an emerging approach for software development. It uses metamodels to define language (or formalism) abstract notation, so separating the abstract syntax and semantics of the language from their different concrete notations. However, metamodelling frameworks lack of a way to specify the semantics of languages, which is usually given in natural language. We claim that the MDE paradigm can gain rigor and preciseness from the integration with formal approaches, and we propose the integration with the ASMs to define a unified methodology for metamodel-based language syntax and semantics definitions.

Keywords: Abstract State Machines, Model-driven Engineering, ASM Metamodel, Metamodelling

Joint work of: Gargantini, Angelo; Riccobene, Elvinia; Scandurra, Patrizia

Extended Abstract: http://drops.dagstuhl.de/opus/volltexte/2006/638

Interactive Algorithms 2006: from Behavioral Theory to Visual Studio?

Dean Rosenzweig (University of Zagreb, HR)

This talk is a work-in-progress report, overviewing the behavioral theory of interactive algorithms, being developed by Y.Gurevich, A.Blass, B.Rossman and the first author, and its implications for the practice of concurrent and distributed programming, being explored by the second author.

We explain and motivate the distinction of ordinary vs. general interactive small-step algorithms, and the splitting of the latter notion to stand-alone and component cases. We show how the notion of wide-step (parallel) algorithm extends to an interactive situation. Each of these classes of algorithms has an appropriate ASM syntax capturing the class exactly.

The derived syntax implies an extension of the current AsmL language to an explicitly interactive version AsmL-I. The work of the second author on the definition and implementation of AsmL-I has met exactly the same problems

as other ongoing attempts at introducing more transparent and efficient paradigms of concurrent/distributed programming. The solutions arrived at seem, by first benchmarks of first prototypes, to be remarkably efficient. Together with the capability of AsmL-I to express communication and finely grained concurrency/parallelism transparently, on a high level of abstraction, this has led a part of the current provisional AsmL-I syntax and pragmatics to be considered for adoption as a general purpose commercial programming tool.

The Mondex Challenge: Machine Checked Proofs for an Electronic Purse

Gerhard Schellhorn (Universität Augsburg, D)

The Mondex case study about the specification and refinement of an electronic purse as defined in [Stepney, Cooper, Woodcock 2000] has recently been proposed as a challenge for formal system-supported verification.

I will report on the successful verification of the case study using the KIV specification and verification system. I will demonstrate that even though the hand-made proofs were elaborated to an enormous level of detail we still could find small errors in the underlying data refinement theory as well as the formal proofs of the case study. I will also show an alternative formalisation of the communication protocol using abstract state machines and explain how the verification fits into our approach of systematic development of E-commerce communication and security protocols.

Full details of the case study (including all proofs and a technical report) are available on http://www.informatik.uni-augsburg.de/swt/projects/mondex.html

Joint work of: Schellhorn, Gerhard; Grandy, Holger; Haneberg, Dominik; Reif, Wolfgang

Typed Abstract State Machines in Data-Intensive Applications

Klaus-Dieter Schewe (Massey University, NZ)

While Abstract State Machines (ASMs) provide a general purpose development method, it is advantageous to provide extensions that ease their use in particular application areas. The talk focuses on such extensions for the benefit of a "refinement calculus" in the area of data-intensive systems. We show that providing typed ASMs helps to exploit the existing logical formalisms used in this area to define a ground model and standard refinement rules. We also show that the extensions do not increase the expressiveness of ASMs, as each typed ASM will be equivalent to an "ordinary" one.

Specification and Verification using JML (Java Modeling Language)

Peter H. Schmitt (Universität Karlsruhe, D)

Using an example from avionics software we will first describe the use of the Java Modeling Langugae (JML). Next we will shortly introduce our formal verification tool, the KeY tool, and its underlying logic, which is a version of Dynamic Logic taylored towards verfication of Java programs. The JML specification are translated into the input language of our interactive theorem prover and proved.

Chunks: component verification in CSP||B

Steve Schneider (University of Surrey, GB)

CSP||B is an approach to combining the process algebra CSP with the formal development method B, enabling the formal description of systems involving both event-oriented and state-oriented aspects of behaviour. The approach provides architectures which enable the application of CSP verification tools and B verification tools to the appropriate parts of the overall description. Previous work has considered how large descriptions can be verified using coarse grained component parts. This talk considers a generalisation of that work so that CSP||B descriptions can be decomposed into finer grained components, _chunks_, which focus on demonstrating the absence of particular divergent behaviour separately.

Keywords: Component based verification; B-Method; CSP; decomposition

Joint work of: Schneider, Steve; Treharne, Helen; Evans, Neil

Full Paper:
http://www.computing.surrey.ac.uk/personal/st/S.Schneider/papers/ifm05.ps

See also: IFM 2005, Integrated Formal Methods, Eindhoven, The Netherlands, 2005

A Light Simulator-Verificator for Timed Abstract State Machines

Anatol Slissenko (Université Paris 12, F)

The talk is a short description of main features of a portable simulator-verificator for reactive timed Abstract State Machines. The time may be continuous or discrete, the time constraints are linear (extendable to polynomial ones), the reaction may be instantaneous or with non deterministic bounded delays. The construction of runs goes on together with the verification of a formula in a First Order Times Logic. There are simple languages for the generation of inputs, for resolving non determinism and for displaying of the results.

UML-B: concept and development

Colin F. Snook (Univ. of Southampton, GB)

Part of the reason for the popularity of the UML in industry may be due to the flexibility of its (initially underspecified) semantics and its extension mechanisms. This approach has allowed specialisations to be defined for different purposes while retaining a broadly familiar notation. UML-B provides a specialisation of the UML for rigorous modelling and specification of systems. It has a formal semantics defined by translation into the B language. This is automated by the tool, U2B. UML-B provides a visual form of B with some additional features such as an object-oriented style lifting of specification parts to a set of instances and a statemachine behavioural specification. UML-B was developed under the MATISSE and PUSSEE projects and continues to be developed as part of the RODIN project. Here we present the original concept and motivation for UML-B, point out some weaknesses and indicate how we plan to improve the UML-B notation tools to overcome this and move towards closer integration with the new Event B platform.

Joint work of: Snook, Colin F.; Butler, Michael J.

ASM Applied to Teaching of Software Engineering

Bernhard Thalheim (Universität Kiel, D)

Software engineering can be considered to be the establishment and the use of sound methods for the efficient construction of large and right-quality software that solve the problems such as users identiy them. It includes the entire development process of the software, the requirements prescription and the domain prescription starting by a description of the application domain. Large software systems can be based on integrated component construction and architecture development. Treatment of quality of systems requires control of quality and check of quality for all products that have been developed during the software development. Thus, SE is based on methods (principles, analysis, construction, technique, tool, and artifact), on methodologies for use of methods and results in documents that are used for both description and analysis. We may base software development on principles (types, functions, relations, algebra, logic), techniques, and tools. Often software engineering is treated in an informal way leaving the foundations aside. Additionally verification and validation is often based on principles that have been developed for the programming in-the-small instead of programming in-the-large or programming in-the-world. We elaborate the common belief that software engineering can be given through a formal basis on the basis of abstract state machines. ASM may escort the entire software engineering process due to their abstraction, refinement, well-foundedness, scalability, flexibility, and universality.

The software engineering course given at Christian-Albrechts-University Kiel also includes management of software engineering products, processes and software quality.

Keywords: Abstract state machines, software engineering, codesign, SiteLang, DistrLang

Extending the use of CSP||B

Helen Treharne (University of Surrey, GB)

CSP||B is an integrated formal method approach which supports compositional verification. Recently, we've been experimenting with using it alongside executable UML to expose weaknesses in concurrent state machines.

The work is very much in its preliminary phase and is part of a continuing industrial collaboration. In the talk I'll aim to show how we've tailored the style of CSP||B specficiations so that they can be applied to reason about UML state machines.

Keywords: CSP, B, modelling, analysis

Computational Modeling and Validation of Probabilistic Aspects of Public Safety and Security Systems

Mona Vajihollahi (Simon Fraser University, CA)

Applying computational modeling and validation techniques to the domain of public safety and security systems can greatly facilitate functional analysis and reasoning about such systems. Abstract state machines provide a suitable semantic framework for formalizing such models and allow for machine assisted inspection of functional requirements. In addition, we contend that, due to the inherent uncertainty in such systems, using probabilistic modeling techniques in combination with probabilistic model checking tools facilitates the analysis of non-functional requirements, such as risk, fault-tolerance, and cost, and provides invaluable feedback for design and calibration of such systems.

Keywords: Computational Modeling and Validation, Abstract State Machine, Probabilistic Modeling and Model Checking, Risk Analysis, Performance Analysis

Joint work of: Glaesser, Uwe; Rastkar, Sarah; Vajihollahi, Mona

The Rodin Platform

Laurent Voisin (ETH Zürich, CH)

ETH Zurich participates to the European Project Rodin, "Rigorous Open Development Environment for Complex Systems". In this context, the ETH team is responsible for developing some kernel tools: a formalism-neutral platform, based on Eclipse, and tools for event-B modelling (static checker, proof obligation generator and prover).

This talk will give an overview of the architecture of the Rodin platform, together with a demonstration of its early prototype, targeted to event-B.

Promoting the use of Formal Methods through Education

Amiram Yehudai (Tel Aviv University, IL)

The use of Formal Methods (FM) in the development of Software is growing slowly. While it is desirable to introduce comprehensive use of FM in industry, it is often easier for an organization to start using FM in a way that complements, rather than replaces, current practices. Academic courses that introduce comprehensive use of FM are often electives, divorced from the main curriculum, and hence do not have enough impact on most students. We propose to promote the introduction of FM as an important thread in general, required Software courses. We relate our experience in teaching Object Oriented Programming courses that emphasize Design by Contract from the start.

Keywords: Design by Contract, Software Engineering education

On the Correctness of Transformations in Compiler Backends

Wolf Zimmermann (Universität Halle-Wittenberg, D)

The talk summarizes the results on the correctenss transformations in compiler back-ends achieved in the DFG-project Verifix. Compiler back-ends transform intermediate representations into target machine code. Back-end generators allow to generate compiler back-ends from a set of transformation rules. The talk focuses on the correctness of these transformation rules and on the correctness of the whole transformation stemming from these transformation rules. The correctness proofs for transformations are simulation proofs showing that the target code is a refinement of the source code.